JavaScript 7

2019년 3월 20일 수요일 오후 2:40

*변수의 scope → 선언한 변수가 유효한 영역

: function의 Scope를 따른다

⇒ Objects are accessible only within declared functions.

(Can include nested function too.)

*함수의 shadowing

: 함수 밖에서 선언되었던 같은 이름의 변수를 함수 안에서 사용하는 경우 함수 밖의 변수는 잠시 가려진다. (shadowing)

• 함수 안에서는 해당 함수에서의 변수를 사용한다.

⇒ 함수 밖 변수의 값은 변하지 않는다.

↔ 함수에서 빠져나오면 해당변수에 다시 접근 가능.

함수 내에서 출력한 Value는 함수내의 변수값을,

함수 밖에서 출력한 Value는 함수밖의 변수의값을 따라간다.

• 한 함수 안에서만 값이 유지되어야하는 변수는 함수안에서 선언.

```
function name{
  var in_function_var;
}
```

• 여러 함수에서 값이 유지되면서 사용되는 변수는 포괄하는 부분에서 선언!

*method. this

: 함수 역시 객체의 속성이 될 수있다. 객체속성의 값으로서 함수가 들어가면 그걸 method라고 한다.

- 일반적인 함수 호출시 해당함수 내부에서 사용된 this는 전역객체에 바인딩!
- method 호출시 this의 값은 인스턴스화된 객체참조.

```
▶ Window {postMessage: f, blur: f, focus: f, close: f, parent: Window, …}
f is called
▼ {name: "object", method: f} ℹ
  ▶ method: f f()
    name: "object"
  ▶ __proto__: Object
f is called
```

<methodTest.js>

```
▼ {name: "object1", method: f, setName: f} ℹ
  ▶ method: f f()
    name: "object1"
  ▶ setName: f setName(name)
  ▶ __proto__: Object
```

```
▼ {name: "object2", setName: f} ℹ
    name: "object2"
  ▶ setName: f setName(name)
  ▶ __proto__: Object
```

<extendedmethodTest.js>

* closure

: 함수와 그 함수의 environment로구성.

```
1  function makeCounterFunction(initVal){
2      var count = initVal;
3      function Increase(){
4          count++;
5          console.log(count);
6      }
7      return Increase; // 함수를 반환하는 함수
8
9  }
10
11 //함수 두 번 호출, 변수에 icrease함수가 저장되도록
12
13 var counter1 = makeCounterFunction(0);
14 var counter2 = makeCounterFunction(10);
15
16 //함수 호출
17 counter1();
18 counter2();
```

makeCounterFunction이 0일때와 10일때!

	(0)	(10)
Closure가 가리키는 함수	function Increase(){ }	
closure의 environment	var count = 0;	var count = 10;

Increase를 이용해 var count에 접근할수있지만 함수 밖에서 var count를 직접접근할 수는 없다.